BNL-108233-2015-CP

Unsupervised Feature Selection on Data Streams

Hao Huang

Presented at the 24th International Conference on Information and Knowledge Management
Melbourne, Australia
October 19-23, 2015

October 2015

Computational Science Center

Brookhaven National Laboratory

U.S. Department of Energy
ASCR

DISCLAIMER

Unsupervised Feature Selection on Data Streams

ABSTRACT

Massive data streams are continuously being generated from sources such as social media, broadcast news, etc., and typically these datapoints lie in high-dimensional spaces (such as the vocabulary space of a language). Timely and accurate feature subset selection in these massive data streams has important applications in model interpretation, computational/storage cost reduction, and generalization enhancement. In this paper, we introduce a novel unsupervised feature selection approach on data streams that selects important features by making only one pass over the data while utilizing limited storage. The proposed algorithm uses ideas from matrix sketching to efficiently maintain a low-rank approximation of the observed data and applies regularized regression on this approximation to identify the important features. We theoretically prove that our algorithm is close to an expensive offline approach based on global singular value decompositions. The experimental results on a variety of text and image datasets demonstrate the excellent ability of our approach to identify important features even in presence of concept drifts and also its efficiency over other popular scalable feature selection algorithms.

1. INTRODUCTION

The *curse of dimensionality* plagues many complex learning tasks. A popular approach for overcoming this problem is by reducing the dimensionality of the feature space as that directly results in a faster computation time. At the same time, it is appealing to have *feature interpretability*, which some of the popular dimensionality reduction methods (e.g., PCA, spectral embeddings) do not possess because of their lack of direct connection to the observed feature space. In our work, we propose a novel approach to unsupervised feature selection, which is the problem of choosing a subset of important (original) features without any label information. The selected feature subset minimizes a very intuitive evaluation criteria while accounting for noise and redundancy. This in turn could lead to better 1) model interpretation, 2) computational efficiency, and 3) generalization ability for the learning task.

One of the most important characteristic for any good feature selection approach is the ability to handle huge volumes of data. Most modern data such as documents, images, multimedia from the web naturally arrives in a streaming fashion. However, detecting an informative feature subset in a large volume of data stream is also a difficult problem due to the following reasons: 1) the data stream could be infinite, so any off-line algorithm that attempts to store the entire stream for analysis will eventually run out of memory, 2) the feature importances change dynamically over time due to *concept-drift*, important features may become insignificant and vice-versa, and 3) for various online applications, it is important to obtain the feature subset in close to real-time.

Although there is considerable amount of previous literature on feature selection both in the batch [4, 16] and online setting [35, 26, 19], none of them handles large volume data stream effectively, given limited memory and CPU time, without any prior knowledge about labels. In practice, streams often contain inherently correlated data [22], so it is possible to reduce a large volume numerical stream into just a handful of hidden basis that compactly describe the key patterns of the data. We exploit this idea to reduce the complexity of streaming feature selection analysis.

Our Techniques. In this paper, we propose a streaming feature selection approach that easily adapts to the concept/topic drift arising in the data stream, and at every timestep provides a feature importance score (weight). Our streaming feature selection algorithm uses ideas from matrix sketching[1] to maintain a *low-rank approximation* of the entire observed data at every timestep, and this approximation is continually updated as new data arrives. For matrix sketching, we modify a recent algorithm (called *Frequent Directions*) proposed by Liberty [17]. The *Frequent Directions* algorithm operates in a streaming model and constructs a sketch matrix using a simple idea of "shrinking" a few orthogonal vectors. However, just the low-rank approximation cannot by itself provide feature weighting. In our research this low-rank approximation is exploited and at every timestep the feature importance score is generated by performing a regression analysis. A regularization is added to prevent overfitting to the data (we explain the choice of regularization in Section 3.3). The idea of using regularized regression for feature selection in an unsupervised setting was recently proposed by Cai *et al.* [4], who empirically showed

[1]A sketch of a matrix Z is another matrix Z' that is much smaller than Z, but still approximates it well [17].

WOODSTOCK '97 El Paso, Texas USA

that it leads to a better choice of features for clustering and classification applications. Their main idea is to obtain feature importance using a regularized regression where the spectral embedding of the dataset is used as the regression target. However, the formulation presented in Cai *et al.* [4] operates in a batch setting and requires access to the entire affinity matrix for the regression step, which is not possible in a streaming setup. Our algorithm on the other hand, requires just *one pass* over the data, which is an essential requirement for any "true" streaming algorithm.

To the best of our knowledge, ours is the first unsupervised feature selection algorithm operating in a true data streaming setting. Our feature selection algorithm is effective and efficient in the following ways:

(a) It is space and time efficient while requiring only one pass over the data. For a stream at time t consisting of n_t ($\geqslant 1$) datapoints in an m-dimensional space, our algorithm requires only $O(mn_t)$ space (linear in the size of the input) and $O(mn_t\ell)$ time, where the sketch matrix is of size $m\times\ell$. In practice, it suffices to set ℓ much smaller compared to m and n_t. Therefore, both the memory and computation requirements are almost linear in the input size (as the input at time t is an $m \times n_t$ matrix).

(b) It easily adapts to unseen patterns on the data stream and provides at every timestep an updated identification of the informative feature subset (i.e., at every timestep t it provides a feature ranking based on all the data that have arrived till time t), which gives it the ability of handling concept drift[2] (related experiments in Section 5.4).

(c) We provide theoretical support for our algorithm (Section 3.4), and show that it has a comparable performance to an expensive offline approach that uses singular value decompositions.

Empirical studies show that our streaming approach is efficient in terms of both space and time, while approaching the performance of popular batch algorithms on a wide array of datasets from both the text and image domains.

2. PRELIMINARIES

Notation. We denote $[n] = 1 : n$. Vectors are always in column-wise fashion and are denoted by boldface letters. For a vector $\mathbf{v}$, $\mathbf{v}^\top$ denotes its transpose and $\|\mathbf{v}\|$ denotes its Euclidean norm. For a vector $(a_1, \ldots, a_m) \in \mathbf{R}^m$, $\mathrm{diag}(a_1, \ldots, a_m) \in \mathbf{R}^{m\times m}$ denotes a diagonal matrix with $a_1, \ldots, a_m$ as its diagonal entries. Let $\mathbb{I}_m$ denote an identity matrix of dimension $m \times m$. We use $\mathrm{rank}(Z)$ to denote the rank of Z. For a matrix $Z \in \mathbf{R}^{m\times n}$, we use $z_{i,j}$ to denote its (i,j)th element. Spectral norm is defined as $\|Z\| = \sup\{\|Z\mathbf{v}\| : \|\mathbf{v}\| = 1\}$. We also use entry-wise norms denoted by $\|Z\|_p$, where $p = 2$ gives (Frobenius norm) $\|Z\|_F^2 = \sum_{ij} z_{i,j}^2$, $p = 1$ gives $\|Z\|_1 = \sum_{i,j} |z_{ij}|$, and $p = \infty$ gives $\|Z\|_\infty = \max_{i,j} |z_{i,j}|$. We use $Z \succeq 0$ if Z is a positive semidefinite (PSD) matrix and $Z \succeq Y$ if $Z - Y \succeq 0$. Given a set of matrices, $Z_1, \ldots, Z_t$, we use the notation $Z_{[t]}$ to denote the matrix obtained by horizontally concatenating $Z_1, \ldots, Z_t$, i.e., $Z_{[t]} = [Z_1|\ldots|Z_t]$.

[2]As we discuss later, in some feature selection applications, one might wish to reweigh the points to emphasize more on the recent points that the older points, which can also be easily handled in our framework.

We use $\textsc{Svd}(Z)$ to denote the singular value decomposition of Z, i.e., $\textsc{Svd}(Z) = U\Sigma V^\top$. Here U is an $m \times m$ orthogonal matrix, Σ is an $m \times n$ diagonal matrix, and V is an $n \times n$ orthogonal matrix. The diagonal entries of Σ, where $\sigma_1 \geqslant \sigma_2 \geqslant \cdots \geqslant \sigma_m$ (given $m \leqslant n$), are known as the singular values of Z. We follow the common convention to list the singular values in non-increasing order. For a symmetric matrix $S \in \mathbf{R}^{m\times m}$, we use $\textsc{Eig}(S)$ to denote its eigenvalue decomposition, i.e., $U\Lambda U^\top = \textsc{Eig}(S)$. Here U is an $m \times m$ orthogonal matrix and Λ is an $m\times m$ diagonal matrix whose (real) entries are $\lambda_1, \ldots, \lambda_m$ are known as the eigenvalues of S (again listed in non-increasing order).

The best rank-k approximation (in both the spectral and Frobenius norm sense) to a matrix $Z \in \mathbf{R}^{m\times n}$ is $Z_{(k)} = \sum_{i=1}^{k} \sigma_i \mathbf{u}_i \mathbf{v}_i^\top$, where σ_i ($i \leqslant k$) are the top-k singular values of Z, with associated left and right singular vectors $\mathbf{u}_i \in \mathbf{R}^m$ and $\mathbf{v}_i \in \mathbf{R}^n$, respectively. We use $\textsc{Svd}_k(Z)$ to denote the truncated singular value decomposition of $Z_{(k)}$, i.e., $Z_{(k)} = \textsc{Svd}_k(Z) = U_{(k)}\Sigma_{(k)}V_{(k)}^\top$. Here $\Sigma_{(k)} = \mathrm{diag}(\sigma_1, \ldots, \sigma_k) \in \mathbf{R}^{k\times k}$, $U_{(k)} = [\mathbf{u}_1, \ldots, \mathbf{u}_k] \in \mathbf{R}^{m\times k}$, and $V_{(k)} = [\mathbf{v}_1, \ldots, \mathbf{v}_k] \in \mathbf{R}^{n\times k}$. The following well-known theorem bounds the approximation error of the best rank-k approximation.

THEOREM 2.1. *[Golub* et al. *[8]] Let $Z \in \mathbf{R}^{m\times n}$ with $n > m$, and let $\sigma_1 \geqslant \cdots \geqslant \sigma_m$ be the singular values of Z. Let $\textsc{Svd}_k(Z) = U_k\Sigma_k V_k^\top$. Then*

$$\min_{\mathrm{rank}(X)\leqslant k} \|Z - X\|_F = \|Z - U_{(k)}\Sigma_{(k)}V_{(k)}^\top\|_F = \sqrt{\sum_{j=k+1}^{m} \sigma_{k+1}^2}.$$

3. FEATURE SELECTION ON STREAMS

In this section, we propose an online feature selection algorithm that operates in a streaming setting. We start by describing the problem of feature selection on data streams. Due to space constraints, we omit detailed proofs here.

3.1 Problem Formulation

We assume that the data items arrive in streams. Let $\{Y_t \in \mathbf{R}^{m\times n_t}, t = 1, 2, \ldots\}$ denote a sequence of streaming items, where Y_t represents the data items introduced at timestep t. Here m is the size of feature space, and $n_t \geqslant 1$ is the number of data items arriving at time t. [3] We normalize Y_t such that each column in Y_t has a unit L_2-norm. Under this setup, feature selection aims at selecting the most informative feature subset based on certain evaluation criteria.

3.2 Our Motivation and Framework

Our main idea is based on maintaining, at each timestep t, a low-rank approximation of all the seen (till time t) data stream. By using a regression analysis on this low rank-matrix, we can weigh each feature with an up-to-date importance score.

In case of unsupervised feature selection, the evaluation criteria for selecting the feature subset is not provided explicitly, and the general idea is that we want to capture the most important characteristics of dataset without loosing too much information. To this end, we perform a spectral decomposition on the affinity matrix to obtain a "flat" embeddings of the datapoints [2]. The intuition being that it

[3]One could consider, a setting where only one point comes at a time (i.e., $n_t = 1$), but by allowing $n_t \geqslant 1$, we allow more flexibility in our setup.

is much easier to capture the global trends of the stream in this embedded space than in the original space. Let $Y_{[t]} = [Y_1|\ldots|Y_t] = U_t\Sigma_t V_t^\top$. Since $Y_{[t]}$ is column-wise normalized to have unit euclidean norm, $Y_{[t]}^\top Y_{[t]} = V_t\Sigma_t^2 V_t^\top$ forms the *cosine affinity matrix* of $Y_{[t]}$.

Cai *et al.* [4] proposed an unsupervised feature selection approach using V_t as the target variable in regression. The resulting regression problem can be stated as:

$$\min_X \|Y_{[t]}^\top X - V_t\|_F^2, \tag{1}$$

where each column in $X \in \mathbf{R}^{m\times n_t}$ contains the combination coefficient for different features in approximating the eigenvectors of $Y_{[t]}^\top Y_{[t]}$ (or equivalently the right singular vectors of $Y_{[t]}$). Note that the solution for (1) is $X_t = U_t\Sigma_t^{-1}$ (assuming all the singular values in Σ_t are non-zero).

Now consider a rank-k approximation of $Y_{[t]}$ as defined by Theorem 2.1 (for an appropriately chosen parameter[4] k), let

$$Y_{[t]_{(k)}} = \textsc{Svd}_k(Y_{[t]}) = U_{t_{(k)}}\Sigma_{t_{(k)}}V_{t_{(k)}}^\top.$$

Given the low-rank approximation of $Y_{[t]}$, instead of using V_t as the regression target, we could use $V_{t_{(k)}}$ in (1). This yields the following (least-squares) regression problem:

$$\min_X \|Y_{[t]}^\top X - V_{t_{(k)}}\|_F^2. \tag{2}$$

Note that the solution for (2) is $X_t = U_{t_{(k)}}\Sigma_{t_{(k)}}^{-1}$ (assuming the top-k singular values of Σ_t are non-zero).

However, simply using (2) may lead to an unstable solution (if the input matrix is ill-conditioned) and also overfitting to the data [4]. Therefore, we add a regularization term, and define:

$$X_t = \text{argmin}_{X=(x_{i,j})}\|Y_{[t]}^\top X - V_{t_{(k)}}\|_F^2 + \alpha\sum_{i,j}|x_{i,j}|^p, \tag{3}$$

where α is the regularization parameter that controls the trade-off between the loss function and the p-norm ($p \in \{1,2\}$). Generally, a regression formulation with L_1- ($p = 1$) and L_2-norm ($p = 2$) regularization are referred to as lasso and ridge regression respectively. The general formula of (3) was first concretized by Cai *et al.* [4] for the case of $p = 1$, who referred to it as *Multi-Cluster Feature Selection (MCFS)*.

Generally speaking, after we obtain $X_t = (x_{t_{i,j}})$ from (3), we can assign feature importance score $\mathbf{w}_t = (w_{t_1},\ldots,w_{t_m}) \in \mathbf{R}^m$ (with the interpretation that the higher the score, the more important the feature is). One of the simplest way is by using the following equation introduced in [4]:

$$\forall i \in [m], w_{t_i} = \max_{1\leqslant h\leqslant k}|x_{t_{i,h}}|, \tag{4}$$

The aforementioned prototype algorithm for feature weighting is documented in Algorithm 1. The subsequent feature selection process can be done by ranking the $\mathbf{w}$ vector (in non-increasing order) and choosing the top-h features with the largest score (given that h features are needed).

Other Kernels. Although we concentrate on the cosine kernel, the above framework can be generalized to other kernel functions. One approach would be to use the random feature map transformations known for all radial-basis function kernels [24]. For instance, a Gaussian kernel can be approximated using random Fourier features [24] such that Gaussian kernel evaluation between a pair of datapoints can be approximated by the Euclidean inner product between the transformed pair. Using this randomized feature map one could, in a streaming fashion, transform all the datapoints, and then work exclusively with these transformed datapoints in a framework similar to detailed above.

Windowed Inputs. In our above problem formulations, all data till time t is used for selecting the top features at time t. However, some applications might require features to be selected based on a rolling window of inputs or by providing higher weights on the recent inputs, etc. Our algorithm could be easily adopted to these scenarios by modifying the matrix sketch construction. For simplicity, we ignore these aspects in this paper.

[4]We defer the discussion on setting of k to later. Readers could think of k as a small number $k \ll \min(m, n_t)$.

Algorithm 1: GenFeatWeight (prototype algorithm for feature weighting)

Input: $Y_{[t]} \in \mathbf{R}^{m\times n_{[t]}}$, k, and $p \in \{1,2\}$
Output: Feature importance score $\mathbf{w}_t \in \mathbf{R}^m$ at time t
$U_{t_{(k)}}\Sigma_{t_{(k)}}V_{t_{(k)}}^\top \leftarrow \textsc{Svd}_k(Y_{[t]})$ (with $\Sigma_{t_{(k)}} = \text{diag}(\sigma_{t_1},\ldots,\sigma_{t_k})$)
$X_t \leftarrow \text{argmin}_{X=(x_{i,j})}\|Y_{[t]}^\top X - V_{t_{(k)}}\|_F^2 + \alpha\sum_{i,j}|x_{i,j}|^p$
$\forall i \in [m]$, $w_{t_i} \leftarrow \max_{1\leqslant h\leqslant k}|x_{t_{i,h}}|$

3.3 Lasso ($p = 1$) vs. Ridge ($p = 2$) Regression

The first efficiency issue in using (3) is due to the regularization type. It is well-known that ridge regression penalizes regression coefficients, rather than accomplishing variable/feature selection, while lasso regression to some extent automatically sets insignificant coefficients to be zero. However, there is no previous analysis in the framework of Algorithm 1 about the performance difference obtained by using either lasso or ridge regressions. In this subsection, we investigate this important topic.

A simpler situation arises when the design matrix of the regression problem consists of *orthogonal columns*. In this case, it is easy to show theoretically that ridge and lasso regression select almost the same features (Corollary 3.2).

Lemma 3.1 (Restated from [40]). *Let $\hat{X}$ denote the simple least squares coefficients, or in other words, $\hat{X} \leftarrow \text{argmin}_X\|Y^\top X - A\|_F^2$. Let $\tilde{X}^R$ and $\tilde{X}^L$ denote the estimators obtained from lasso and ridge regressions, respectively. If $Y^\top$ has orthogonal columns, then $\hat{X} = YA$, and we have the following closed-form expressions:*

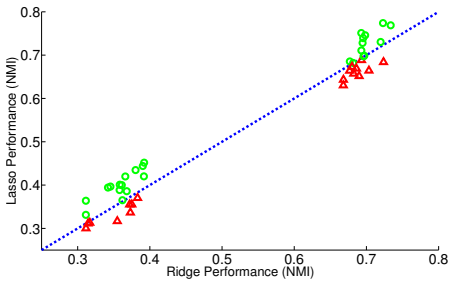
$$\forall i \in [m], j \in [k],\ \tilde{X}^L_{i,j} = \text{sign}(\hat{X}_{i,j})\{|\hat{X}_{i,j}| - \alpha/2\}_+,$$
$$\tilde{X}^R = \hat{X}/(1+\alpha),$$

where for any scalar z, z_+ denotes the positive part, which is z if $z > 0$ and 0 otherwise.

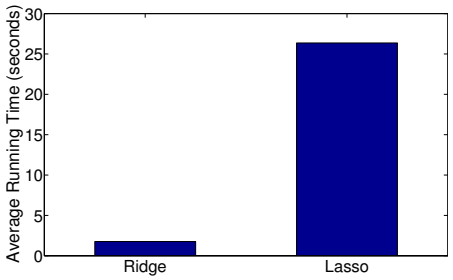
Let $\mathbf{w}^R$ and $\mathbf{w}^L$ be the feature importance score calculated from $\tilde{X}^R$ and $\tilde{X}^L$ using (4). The following corollary follows because $\tilde{X}^L$ and $\tilde{X}^R$ are based on thresholding or scaling $\hat{X}$.

Corollary 3.2. *If $\mathbf{w}^L$ has h non-zero weight features, then under the assumption of Lemma 3.1, the ranking of the top-h features in $\mathbf{w}^L$ coincides with the ranking of the top-h features in $\mathbf{w}^R$.*

The above corollary implies, under the column orthogonality constraint on the design matrix, the performance of ridge and lasso regression in Algorithm 1 (especially, for the important features), are almost the same. Therefore, we can potentially use the more computationally cheaper regularization without significant loss in performance.



(a)



(b)

Figure 1: Performance comparison between ridge ($p = 2$) and lasso ($p = 1$) regression in the framework of (3). The top 1000 features are picked using (4). In Figure 1(a), red triangles show the tests when ridge outperforms lasso, while green circles show the tests when lasso outperforms ridge. Although in general lasso slightly outperforms ridge regression, the running time test in Figure 1(b) shows that ridge regression is far more efficient than lasso. The parameter α was manually tuned here (we will further analyze the setting of α in Section 5).

Note that the orthogonality of the design matrix is a very rigid constraint. However, as we will see later (Algorithm StreamFeatWeight) in our setting, the design matrix will be of form $(U\Sigma)^\top$, where U is a matrix with orthogonal columns and Σ is a diagonal matrix (in our case, matrix of singular values). Since such a design matrix may not be *too far* from having orthonormal columns, we have a much higher chance of observing similar results from lasso and ridge regression (as in Corollary 3.2) than in a general regression setting.[5] We conducted the following experiments to verify this hypothesis. We randomly sampled 45 datapoints from three text datasets ("20 Newsgroup", "RCV1", and "Reuters21578", refer to Section 5.1 for details about these datasets). For both cases of $p = 1$ and $p = 2$ in Algorithm GenFeatWeight, we generated the top 1000 feature set from $\mathbf{w}^L$ and $\mathbf{w}^R$ respectively. We then used a simple K-means algorithm on these selected features to evaluate their effectiveness in identifying the (true) document classes (the reasoning behind performing such an evaluation is explained in Section 5). Figure 1(a) shows the clustering result under the Normalized Mutual Information (NMI) measure, and the results suggest that lasso and ridge regression have comparable performance in general. However, in terms of running time (Figure 1(b)), ridge regression is substantially better than lasso regression.[6] Since our goal is to handle large streaming datasets, ridge regression appears as a better choice[7], since it obtains results very similar to lasso regression in the framework of Algorithm GenFeatWeight, but with far lesser running time. Therefore, we concentrate on ridge regression from here on.

The simple closed form solution for (3) with $p = 2$ is as shown in the following lemma.

Lemma 3.3. *Consider the ridge regression solution,*

$$X_t = argmin_{X=(x_{i,j})} \|Y_{[t]}^\top X - V_{t_{(k)}}\|_F^2 + \alpha \sum_{i,j} x_{i,j}^2.$$

Then we have the following:

$$X_t = U_{t_{(k)}} diag(\sigma_{t_1}/(\sigma_{t_1}^2 + \alpha), \ldots, \sigma_{t_k}/(\sigma_{t_k}^2 + \alpha)),$$

where $\sigma_{t_1}, \ldots, \sigma_{t_k}$ are the top-k singular values in $Y_{[t]_{(k)}}$.

Even though Algorithm GenFeatWeight is quite simple, in a streaming environment the number of data items in $Y_{[t]}$ could become extremely large, which could lead to both computational and memory bottlenecks in running the algorithm. For example, the computational complexity of computing a truncated rank-k SVD is $O(mn_{[t]}k)$ [8] (given $Y_{[t]} \in R^{m \times n_{[t]}}$), which is prohibitive when the number of columns in $Y_{[t]}$ becomes very large ($n_{[t]} \to \infty$). Our goal in the next section is to propose and analyze an efficient approach that has similar effectiveness as Algorithm GenFeatWeight in identifying top features but does so by utilizing limited storage and just one pass over the data in a streaming setting.

3.4 Matrix Sketching for Feature Weighting

As mentioned above the main bottleneck in Algorithm GenFeatWeight is in generating a low-rank approximation of $Y_{[t]}$. To overcome this problem, we propose an approach based on matrix sketching that we outline next.

In his recent paper, Liberty [17] showed that by adapting the Misra-Gries approach for approximating frequency counts in a stream [20], one could obtain additive error bounds for matrix sketching. More formally, in the setting of [17], the input is a matrix $Z \in \mathbf{R}^{p \times d}$. In each step, one row of Z is processed by the algorithm (called *Frequent Directions*) in a streaming fashion, and the algorithm iteratively updates a matrix $Q \in \mathbf{R}^{q \times d}$ ($q \ll p$) such that for any unit vector $\mathbf{x} \in \mathbf{R}^d$, $\|Z\mathbf{x}\|^2 - \|Q\mathbf{x}\|^2 \leqslant 2\|Z\|_F^2/q$.

Recently, Ghashami and Philips [7], reanalyzed the *Frequent Directions* algorithm of Liberty [17], to show that it provides relative error bounds for low-rank matrix approximation. Instead of Q, their algorithm return Q_k (a rank-

[5]Note that if U has orthogonal columns and Σ is a diagonal matrix, then $(U\Sigma)^\top$ has orthogonal rows but depending on the singular values in Σ, the columns in $(U\Sigma)^\top$ may not exactly be orthogonal.

[6]For lasso regression, we use the algorithm proposed by Cai *et al.* [4].

[7]Again note that this is not to be misconstrued as a general statement on the effectiveness of ridge vs. lasso regression in other settings.

k approximation of Q) and their main result shows that $\|Z\|_F^2 - \|Q_k\|_F^2 \leqslant q/(q-k) \cdot \|Z - Z_k\|_F^2$.

Our approach for constructing a low-rank approximation of $Y_{[t]}$ (outlined between Steps 1-4 in Algorithm StreamFeatWeight) is based on extending the *Frequent Directions* algorithm of [17] to a more general setting where in every timestep, we add $n_t \geqslant 1$ new columns.[8] As in *Frequent Directions*, our algorithm requires just one pass over the data stream. Here, $B_t \in \mathbf{R}^{m \times \ell}$ is the matrix sketch. The parameter $\ell \geqslant k$, but is generally much smaller than m or n_t. We discuss more on the setting of ℓ later. The Step 5 in Algorithm StreamFeatWeight is obtained by using Lemma 3.3 to solve the following ridge regression problem (note that the SVD of $B_t = \tilde{U}_{t_{(\ell)}} \breve{\Sigma}_{t_{(\ell)}} \mathbb{I}_\ell$, so the identity matrix $\mathbb{I}_\ell$ represents the right singular vectors of B_t):

$$\tilde{X}_t = \text{argmin}_{X=(x_{i,j})} \|B_t^\top X - [\mathbf{e}_1, \ldots, \mathbf{e}_k]\| + \alpha \sum_{i,j} x_{i,j}^2, \quad (5)$$

where $\mathbf{e}_i \in \mathbf{R}^\ell$ is a vector with 1 at location i, and 0 elsewhere (i.e., standard basis vector).

Algorithm 2: StreamFeatWeight (streaming update of feature weights at time t)

Input: $Y_t \in \mathbf{R}^{m \times n_t}$, $B_{t-1} \in \mathbf{R}^{m \times \ell}$, and $\alpha \in R$
Output: Feature importance score $\tilde{\mathbf{w}}_t \in \mathbf{R}^m$ and matrix sketch B_t at time t

1 $C_t \leftarrow [B_{t-1} | Y_t]$
2 $\tilde{U}_{t_{(\ell)}} \tilde{\Sigma}_{t_{(\ell)}} \tilde{V}_{t_{(\ell)}}^\top \leftarrow \text{Svd}_\ell(C_t)$ (with $\tilde{\Sigma}_{t_{(\ell)}} = \text{diag}(\tilde{\sigma}_{t_1}, \ldots, \tilde{\sigma}_{t_\ell})$)
3 $\breve{\Sigma}_{t_{(\ell)}} \leftarrow \text{diag}\left(\sqrt{\tilde{\sigma}_{t_1}^2 - \tilde{\sigma}_{t_\ell}^2}, \sqrt{\tilde{\sigma}_{t_2}^2 - \tilde{\sigma}_{t_\ell}^2}, \ldots, \sqrt{\tilde{\sigma}_{t_{\ell-1}}^2 - \tilde{\sigma}_{t_\ell}^2}, 0\right)$
4 $B_t \leftarrow \tilde{U}_{t_{(\ell)}} \breve{\Sigma}_{t_{(\ell)}}$
5 $\tilde{D}_{t_{(k)}} \leftarrow \text{diag}(\breve{\sigma}_{t_1}/(\breve{\sigma}_{t_1}^2 + \alpha), \ldots, \breve{\sigma}_{t_k}/(\breve{\sigma}_{t_k}^2 + \alpha))$ ($\breve{\sigma}_{t_i}$ is the ith diagonal element in $\breve{\Sigma}_{t_{(\ell)}}$)
6 $\tilde{X}_t \leftarrow \tilde{U}_{t_{(k)}} \tilde{D}_{t_{(k)}}$
7 $\forall i \in [m], \tilde{w}_{t_i} \leftarrow \max_{1 \leqslant p \leqslant k} |\tilde{x}_{t_{i,p}}|$, where $\tilde{X}_t = (\tilde{x}_{t_{i,p}})$

At any time t, the running time of Algorithm StreamFeatWeight is $O(\max\{mn_t\ell, m\ell^2\})$ ($= O(mn_t\ell)$ if we assume $\ell \leqslant n_t$) by using power-iteration or rank-revealing QR decomposition for SVD [8] in Step 2. This computational complexity is much smaller than the $O(mn_{[t]}k)$ time complexity of Algorithm StreamFeatWeight (since $n_t\ell \ll n_{[t]}$). Between iterations, the algorithm only maintains the B_t matrix which takes $O(m\ell)$ storage. The overall space complexity of Algorithm StreamFeatWeight is linear in the size of the input (i.e., $O(mn_t)$) at every time t, compared to Algorithm GenFeatWeight for which, at time t, the space complexity is $O(mn_{[t]})$

The major focus of the rest of this section is to provide theoretical support for Algorithm StreamFeatWeight, by showing that the $\tilde{\mathbf{w}}_t$ from Algorithm StreamFeatWeight is a good approximation of $\mathbf{w}_t$ obtained from Algorithm GenFeatWeight (with $p = 2$).

[8] A similar sketching based low-rank matrix approximation approach was recently used in an entirely different context of anomaly detection by Hao and Kasiviswanathan [11]

3.4.1 Theoretical Comparison (Bounding $\|\mathbf{w}_t - \tilde{\mathbf{w}}_t\|$)

We start with observation that,

$$\begin{aligned}
\|\mathbf{w}_t - \tilde{\mathbf{w}}_t\| &\leqslant \|U_{t_{(k)}} D_{t_{(k)}} - \tilde{U}_{t_{(k)}} \tilde{D}_{t_{(k)}}\|_F \\
&= \|U_{t_{(k)}} D_{t_{(k)}} - \tilde{U}_{t_{(k)}} D_{t_{(k)}} + \tilde{U}_{t_{(k)}} D_{t_{(k)}} - \tilde{U}_{t_{(k)}} \tilde{D}_{t_{(k)}}\|_F \\
&\leqslant \|U_{t_{(k)}} D_{t_{(k)}} - \tilde{U}_{t_{(k)}} D_{t_{(k)}}\|_F + \|\tilde{U}_{t_{(k)}} D_{t_{(k)}} - \tilde{U}_{t_{(k)}} \tilde{D}_{t_{(k)}}\|_F \\
&\leqslant \|U_{t_{(k)}} - \tilde{U}_{t_{(k)}}\|_F \|D_{t_{(k)}}\| + \|\tilde{U}_{t_{(k)}}\| \|D_{t_{(k)}} - \tilde{D}_{t_{(k)}}\|_F \\
&\leqslant \|U_{t_{(k)}} - \tilde{U}_{t_{(k)}}\|_F \|D_{t_{(k)}}\| + \sqrt{k} \|D_{t_{(k)}} - \tilde{D}_{t_{(k)}}\|_\infty. \quad (6)
\end{aligned}$$

Therefore, a bound on $\|\mathbf{w}_t - \tilde{\mathbf{w}}_t\|$ follows from respective bounds on $\|U_{t_{(k)}} - \tilde{U}_{t_{(k)}}\|_F$ and $\|D_{t_{(k)}} - \tilde{D}_{t_{(k)}}\|_\infty$. Note that since the columns in $\tilde{U}_{t_{(k)}}$ are orthonormal, $\|\tilde{U}_{t_{(k)}}\| \leqslant 1$.

Bounding $\|U_{t_{(k)}} - \tilde{U}_{t_{(k)}}\|_F$. Here we use a recent result by Huang and Kasiviswanathan [11], who established an upper bound on $\|U_{t_{(k)}} - \tilde{U}_{t_{(k)}}\|_F$ by modifying the analysis of *Frequent Directions* by Ghashami and Philips [7] and combining it with some recent matrix perturbation results. To formally state their result we need few more definitions. Let

$$\kappa = \kappa_k(Y_{[t]}) = \sigma_{t_1}/\sigma_{t_k},$$

where σ_{t_i} is the ith singular value of $Y_{[t]}$.

1. Define Γ_a as,

$$\Gamma_a = \frac{\kappa^2 \|Y_{[t]_{(k)}}\|_F^2 - \|B_{t_{(k)}}\|_F^2}{\|Y_{[t]_{(k)}}\|_F^2 - \|B_{t_{(k)}}\|_F^2}. \quad (7)$$

It is easy to establish that for all t, $\|Y_{[t]_{(k)}}\|_F^2 \geqslant \|B_{t_{(k)}}\|_F^2$ (using an analysis from [11]), and by definition $\kappa \geqslant 1$, therefore $\Gamma_a \geqslant 1$. Furthermore, for small k's (as in our setting), typically κ is bounded, yielding $\Gamma_a = O(1)$.

2. Define Γ_b as,

$$\Gamma_b = 1 + \frac{2}{\kappa^2 - \|B_t\|^2/\|Y_{[t]}\|^2}. \quad (8)$$

Again it is easy to establish that $Y_{[t]} Y_{[t]}^\top \succeq B_t B_t^\top$ [17], and therefore, $\|B_t\|^2 \leqslant \|Y_{[t]}\|^2$. Typically κ is also bounded away from 1, yielding $\Gamma_b = O(1)$.

Proposition 3.4 (Huang and Kasiviswanathan [11]). *Let λ_i denote the ith eigenvalue of $Y_{[t]} Y_{[t]}^\top$ and $L = \min_{i \neq j} |\lambda_i - \lambda_j| > 0$. If*

$$\ell = \Omega\left(\frac{\sqrt{m}\kappa^2 \|Y_{[t]}\|^2 \Gamma_a \Gamma_b k \|Y_{[t]} - Y_{[t]_{(k)}}\|_F^2}{L^2}\right),$$

for Γ_a, Γ_b defined in (7), (8) respectively, then

$$\|U_{t_{(k)}} - \tilde{U}_{t_{(k)}}\|_F \leqslant \frac{\sqrt{2}L}{\sqrt{L + 8\kappa^2 \|Y_{[t]}\|^2} \sqrt[4]{L^2 + 16\kappa^4 \|Y_{[t]}\|^4}}.$$

Remark: For small k's, and assuming $1 < \kappa \leqslant O(1)$ (implying $\Gamma_a = O(1)$ and $\Gamma_b = O(1)$), the above bound on ℓ could be simplified to,

$$\ell = \Omega\left(\frac{\sqrt{m}\|Y_{[t]}\|^2 \|Y_{[t]} - Y_{[t]_{(k)}}\|_F^2}{L^2}\right).$$

The assumption of $L > 0$ is also something that is commonly satisfied in practice, especially if m is reasonably smaller than the number of data items in $Y_{[t]}$.

Bounding $\|\mathbf{D}_{t_{(k)}} - \tilde{\mathbf{D}}_{t_{(k)}}\|_\infty$. Let σ_{t_i} and $\breve{\sigma}_{t_i}$ be the ith singular value of $Y_{[t]}$ and B_t respectively. We have,

$$\|D_{t_{(k)}} - \tilde{D}_{t_{(k)}}\|_\infty = \max_{i\in[k]} \left| \frac{\sigma_{t_i}}{{\sigma_{t_i}}^2 + \alpha} - \frac{\breve{\sigma}_{t_i}}{\breve{\sigma}_{t_i}^2 + \alpha} \right| \leqslant \max_{i\in[k]} \left(\frac{|\sigma_{t_i} - \breve{\sigma}_{t_i}|}{\breve{\sigma}_{t_i}^2 + \alpha} \right).$$

A standard application of Weyl's inequality [8], along with a bound on $\|\kappa^2 Y_{[t]} Y_{[t]}^\top - B_t B_t^\top\|$ provides the following proposition.

PROPOSITION 3.5.

$$\|D_{t_{(k)}} - \tilde{D}_{t_{(k)}}\|_\infty \leqslant \frac{\|Y_{[t]} - Y_{[t]_{(k)}}\|_F}{\breve{\sigma}_{t_k}^2 + \alpha} \sqrt{\frac{\Gamma_a \Gamma_b k}{\ell - k}}.$$

Putting it all Together (Bounding $\|\mathbf{w}_t - \tilde{\mathbf{w}}_t\|$). The following theorem follows by combining (6) with Propositions 3.4 and 3.5.

THEOREM 3.1. *Let $Y_1, \ldots, Y_t$ be a sequence of matrices with $Y_{[t]} = [Y_1|\ldots|Y_t]$. Let $Y_{[t]_{(k)}} = U_{t_{(k)}} \Sigma_{t_{(k)}} V_{t_{(k)}}^\top$ be the best rank-k approximation of $Y_{[t]}$. Let σ_{t_k} and $\breve{\sigma}_{t_k}$ be the kth singular value of $Y_{[t]}$ and B_t respectively. Then $\mathbf{w}_t$ (generated by the Algorithm* GENFEATWEIGHT*) and $\tilde{\mathbf{w}}_t$ (generated by Algorithm* STREAMFEATWEIGHT*), under conditions from Proposition 3.4, satisfy:*

$$\|\mathbf{w}_t - \tilde{\mathbf{w}}_t\| \leqslant \frac{k\|Y_{[t]} - Y_{[t]_{(k)}}\|_F}{\breve{\sigma}_{t_k}^2 + \alpha} \sqrt{\frac{\Gamma_a \Gamma_b}{\ell - k}} + \frac{\sigma_{t_k}}{\sigma_{t_k}^2 + \alpha} \frac{\sqrt{2}L}{\sqrt{L + 8\kappa^2 \|Y_{[t]}\|^2 \sqrt[4]{L^2 + 16\kappa^4 \|Y_{[t]}\|^4}}}.$$

The above theorem shows that, under reasonable assumptions and setting of ℓ, both Algorithms GENFEATWEIGHT and STREAMFEATWEIGHT generate very identical feature vector weights at every timestep t. But as we discussed earlier, Algorithm STREAMFEATWEIGHT is far more efficient both in space and time consumptions.

A point to note is that the Algorithm STREAMFEATWEIGHT can be used *with any value* of ℓ, the above bound on ℓ only guarantees that its feature selection results are similar to that of Algorithm GENFEATWEIGHT.

3.5 Discussion on Normalization

We generate $V_{t_{(k)}}$ using a cosine affinity matrix from L_2-norm column normalized $Y_{[t]}$. If we follow the same construction with a slightly different normalization, $Y_{[t]}$ to be zero mean and unit variance, then we obtain as regression target, the principal components (PCA) of $Y_{[t]}$. Each principal component captures different view of Pearson correlation coefficient matrix, and these principal components might be another good choice for the regression target. However, it is unclear how to construct this zero mean and unit variance normalization of $Y_{[t]}$ in a strict streaming fashion which is desired in this paper. The works of [18, 12] use $N^{-1} Y_{[t]}^\top Y_{[t]}$ for building random walk normalized cosine affinity matrix where the diagonal matrix $N_{i,i} = \sum_j (Y_{[t]}^\top Y_{[t]})_{i,j}$. Since we did not use the random walk normalization, $V_{t_{(k)}}$ is not approximating the normalized graph cut any more. However, in our case, $V_{t_{(k)}}$ is not directly used for clustering, but just an intermediate step to select the important feature subset. In our experiments (Section 5), we achieve quite similar results to the MCFS approach [4], where the affinity matrix is normalized in a batch setting.

4. RELATED WORK

We now justify the utility of our proposed approach by briefly comparing it with a few existing methods.

Our basic idea is to use regression analysis for feature selection. Many feature selection algorithms based on this idea have been proposed in the past decade. These algorithms operate by minimizing some appropriately defined objective function. Weston *et al.* [30] added an ℓ_0-norm constraint on the solution to enforce sparsity, which naturally leads to a natural variable (feature) selection. But minimizing with ℓ_0-regularization is NP-hard, therefore ℓ_1-norm, as a convex relaxation to ℓ_0-norm, was utilized in [31, 4]. Other norms on the regularization term such as ℓ_2-norm [6] and $\ell_{2,1}$-norm [37, 10, 16] have also been explored for feature selection. One of the latest work in this area, proposed by Zhu *et al.* [39], performs feature selection by transfering models learnt on external (auxiliary) data sources, but it requires the dimensionality of target data to be high and the number of datapoints to be small, which usually is not the case in data streaming. Few other recently proposed approaches in this area include [36, 13]. Although these methods are effective and robust to some degree, they are extremely inefficient, in both time and space, to be applicable in a streaming setting.

Feature selection algorithms operating in a online setting were proposed in [35, 26, 19], but they all require multiple passes over the data to converge to a stable model, and hence are not *pass-efficient*. Few other efficient feature selection methods such as [9, 33, 34] seem not well-suited to operate in a streaming environment. In this paper, we propose a streaming algorithm that at every timestep efficiently assigns each feature an importance score (weight) that can be subsequently used to rank (or select) the (top) features.

Also in the streaming setting, the approach of *projected clustering* [1, 21] can be viewed as a technique for "local" feature selection. The idea here is to have each cluster specific to a particular feature subset that optimizes a quality criterion for that cluster. However, the feature subsets could be quite different across different clusters, therefore it leads to a complicated interpretation of clustering. Moreover, it is also very difficult to compare different clusters since their optimized subspaces are not in the same domain. On the other hand, our proposed approach provides a single comprehensive feature subset that covers all the clusters. Thereby, it gives an easy interpretation for clustering different classes.

In a somewhat orthogonal setting, online feature selection operating on *feature streams*[9] (instead of data streams as considered in this paper) have been investigated in [23, 38, 32, 15, 29].

5. EXPERIMENTAL ANALYSIS

In this section, we experimentally demonstrate that our proposed Algorithm STREAMFEATWEIGHT is highly scalable, while still providing almost similar quantitative results to other expensive batch feature selection approaches.

[9]Roughly, in this setting, feature vectors are streamed over time, e.g., one new feature is introduced at every timestep t.

	Dataset	#instances	#features	#clusters
1	Reuter21578	8,293	18,933	65
2	TDT2	9,394	36,771	30
3	20Newsgroup	18,846	26,214	20
4	RCV1	193,844	47,236	103
5	USPS	9,298	256	10
6	MNIST	70,000	784	10
7	Tiny	1,000,000	3,072	75,062

Table 1: Statistics of the experimental datasets.

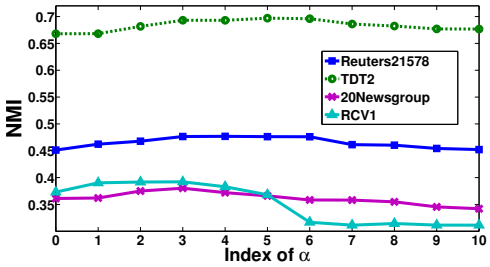


Figure 2: The effect of α on four text datasets. For each dataset, we randomly generate 30 subsets and record the average K-means (NMI) result on the top 1000 features. The x-axis is the index i of $\alpha = 2^i\sigma_k$ (where k is number of clusters in each dataset). It can be observed that the NMI results show smooth changes across different α, and $i = 3, 4, 5$ are reasonable choices.

5.1 Experiment Setup

It would be the best to evaluate feature selection results based on ground truth feature importance. But in real world applications, we cannot easily find such ground truth because: 1) it is highly subjective to select candidate features because there are many similar features/terms, and 2) feature selection is typically an intermediate step for the rest of data analysis pipeline. However, we do have many datasets with ground truth cluster labels. This can be utilized to evaluate the quality of selected feature subset, by performing an unsupervised clustering on the feature-reduced dataset. If the selected feature subset is "good", then clustering the data restricted to just this subset of features should yield a "good" clustering result. Therefore, we evaluate the unsupervised feature selection algorithms by performing an unsupervised K-means clustering on the selected feature space. We used the popular Normalized Mutual Information (NMI) as our evaluation metric (detailed definition of NMI can be referred to [27]). All the experiments were run on an Intel(R) Xeon(R) CPU X5650 2.67GHz processor with 128GB memory.

Baselines. From now on, we refer to Algorithm StreamFeatWeight as **FSDS** (Feature Selection on Data Streams). We chose the following unsupervised feature selection approaches as baseline methods: Multi-Cluster Feature Selection (**MCFS**) [4], **LaplacianScore** [9], and Algorithm GenFeatWeight with $p = 2$ (henceforth, referred to as **GFW-p2**). MCFS (based on lasso regression) and LaplacianScore (based on finding local manifold structure) are both batch feature selection algorithms and were selected for comparison because they capture the essence of two popular approaches to feature selection.

Since we use clustering to measure the performance of feature selection, we also included for comparison the classical K-means (**Kmeans**) and a recent streaming variant of K-means (**StreamKM**) [25]. Both these K-means algorithms are operated on the whole feature set (unlike other compared approaches). To the best of our knowledge there are no other streaming unsupervised feature selection algorithms.

Datasets and Preprocessing. We evaluated the above algorithms on four popular text datasets (Reuter21578, TDT2, 20Newsgroup, and RCV1) and three image datasets (USPS, MNIST, and Tiny), whose statistics are summarized in Table 5.1. 20Newsgroup is a balanced dataset that covers 20 news topics. Reuters21578, TDT2, RCV1 are unbalanced datasets with quite different sizes of clusters. All these datasets can be found in [3]. Both the USPS and MNIST datasets have 10 classes of handwritten digits. Tiny is a large web-image collection for non-parametric object and scene recognition (downloaded from [28]). Among 80 million images, we randomly selected 1 million images and evaluated the result on $60,000$ labeled images that cover 100 classes (from [14]). We directly performed experiments on the Tiny images with all the $3,072$ raw features, which are 32×32 color images in RGB color channels.

MCFS and LaplacianScore algorithms are space and time inefficient due to computations of normalized Laplacian matrix and eigenvalue decomposition (also solving lasso regression for MCFS). We cannot evaluate such algorithms on large benchmark datasets. Therefore the majority of datasets were selected to compare the effectiveness of all the baselines. For the three image datasets, we used the approximation of Gaussian kernel using random feature maps [24]. For every timestep t, we used the same random projection basis and offset setting (refer to [12] for more details about implementing these random feature maps).

Parameter Settings. The number of selected features, h, was set from 200 to 2400 (in increments of 200) for text datasets, and from 25 to 200 (in increments of 25) for image datasets (since the number of meaningful features in raw images is usually small). In our streaming setting, the number of singular vectors k was set to be the same as the number of clusters in the dataset, which was assumed to be *a priori* known as in prior works [9, 4], and the size of each data stream (n_t's) as 1000 (we will further analyze stability against n_t in Section 5.4). Our proposed algorithm has two specific parameters: the size of matrix sketch ℓ (which is set as the square root of the feature size as suggested by the analysis in Section 3.4) and the regularization parameter α. There are prior works in deciding the best regularization parameter α [5]. However, given the target problem is unsupervised and the dataset is big, we performed the following experiments on sampled datasets to select α. We randomly generated 30 subsets from four text datasets respectively, and evaluated the average quality of the top-1000 selected features using Lemma 3.3 with various values of α. Since the singular value distributions (σ_j's) of most datasets usually decay rapidly, we set α as $2^i\sigma_k$ for $i \in \mathbf{R}$. The average NMI result is presented in Figure 2. It can be seen that the best results appear when i is around 3 to 5. Therefore we set $i = 3$ by default, and set $\alpha = 2^3\sigma_k$ for all our experiments. For the random feature maps using Gaussian kernel, we set bandwidth-scale as 5000 and the projected dimension as $\lceil n/k \rceil$ (following [12]).

For MCFS and LaplacianScore, we followed [4, 9] for their respective parameter settings. For the NMI evaluation step, we utilized the standard *within-cluster sum of squares* K-means (with 100 inner loops and 100 outer loops) to obtain stable cluster assignments.

5.2 General Performance Comparison

We make the following observations based on Figure 3:

(1) Compared with Kmeans/StreamKM on the whole features space, feature selection can indeed improve clustering performance on these high dimensional datasets. This is an argument in favor of performing feature selection (similar observations have been made elsewhere).

(2) In general, the regression-based algorithms (FSDS, GFW-p2, and MCFS) perform much better than the LaplacianScore algorithm. It is because LaplacianScore evaluates features individually, so that the selected feature subset may come from similar global patterns. On the other hand, the other three algorithms have more comprehensive views due to their use of regression-based feature selection.

(3) GFW-p2 has very comparable result with MCFS (which can be seen as evidence supporting the lasso vs. ridge argument in Section 3.3), although the latter performs better than the former in some specific spots (e.g., with smaller number of features in the TDT2 dataset).

(4) On average, FSDS achieves more than 99% NMI of GFW-p2 on the text datasets. It confirms our theoretical proof (Theorem 3.1) that feature weight vectors produced by Algorithms GenFeatWeight and StreamFeatWeight are close to each other.

(5) Typically, FSDS achieves about 97% ~ 99% NMI of MCFS on the text datasets. This observation shows that in our problem setting (Sections 3.2 and 3.3), a streaming ridge-based method is capable of obtaining similar performance as that of a batch lasso-based method (MCFS). The FSDS performs worse than MCFS when the number of features is small (TDT2 dataset). It could probably be because the regression target of MCFS that comes from normalized spectral analysis may boost the quality of the small number of the selected features. However, if the number of features is large enough, FSDS and MCFS have very comparable performances.

(6) For the experiments on the image datasets (with approximated Gaussian kernel), FSDS has similar or even better performance than MCFS (Figure 3(e) and 3(f)).

(7) On large datasets, such as RCV1 (Figure 3(d)) and Tiny (Figure 3(g)), MCFS and LaplacianScore algorithms ran out of memory since they require constructing the affinity matrix (which takes $O(n^2)$ space). Memory troubles also prevented GFW-p2 from completion on the Tiny dataset.

5.3 Scalability Comparison

Figure 4 shows the scalability comparison between the feature selection algorithms using Tiny dataset (k is set as 10 here). FSDS requires just few minutes to generate feature importance scores on a dataset with a million points. We observed that FSDS is on average about 10 times faster than LaplacianScore and about 50 times faster than MCFS. Also FSDS outperforms GFW-p2 when the dataset size is above $10,000$ and the difference between their running times will grow as the dataset size increases.

On the 20Newsgroup dataset, FSDS takes about 23 seconds, and is about 3, 35, and 100 times faster than GFW-p2, LaplacianScore, and MCFS respectively. Similarly, on the MNIST dataset, FSDS takes about 4 seconds, and is about 4, 64, and 400 times faster than GFW-p2, LaplacianScore, and MCFS respectively

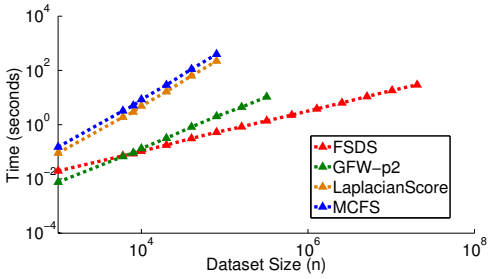


Figure 4: Scalability experiments on the Tiny dataset. Except FSDS (our proposed approach), none of the other compared approaches could scale beyond $\approx 10^5$ points (failing because of their extremely high memory overhead).

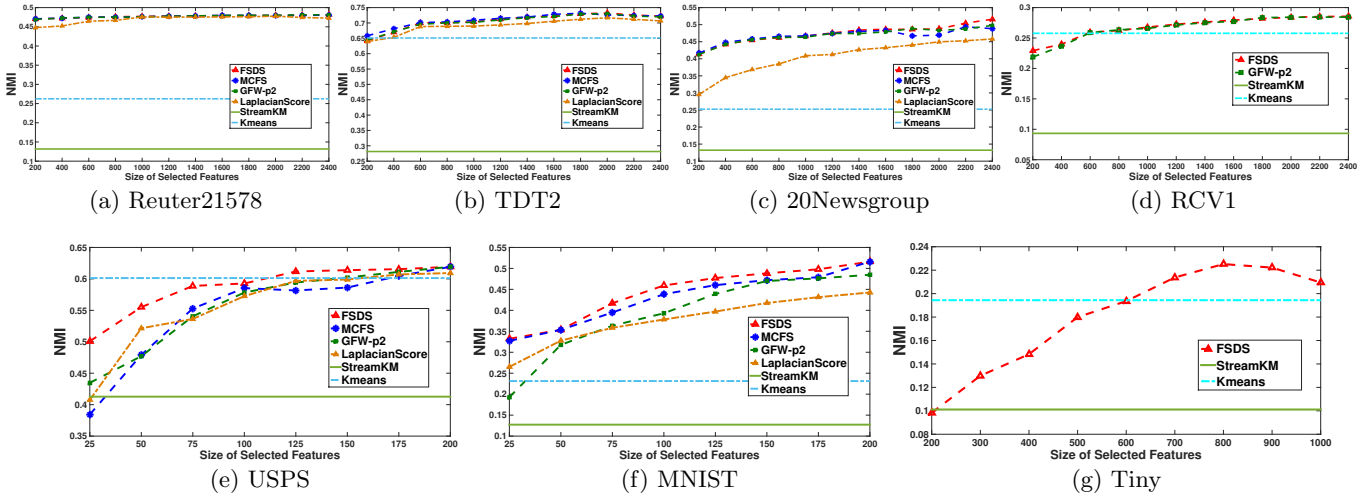
5.4 Stability under Concept Drift, Batch Sizes

It is well-known that streaming algorithms are generally sensitive to the order of data, or concept drift. To test the performance of FSDS in such scenarios, we used the data stream sorted by timestamps as input. The performance of FSDS in this realistic testing environment is shown in Figure 5, with different sizes of feature set. We also compare against a scheme where we just use a static feature subset ($\#f = 200$) without adapting to concept drift. This static feature subset was determined by FSDS using only the first $2,000$ samples. For the two unbalanced datasets Reuters21578 and TDT2, the larger clusters appear in the very beginning. Therefore, initially the approach based on static feature subset performs quite close to FSDS. However, as time goes on and concept drift becomes more prominent, FSDS continues maintaining a good stable performance across all the three datasets, which demonstrates that FSDS is capable of quickly adapting to concept drift.

FSDS tests across different batch sizes (n_t's) and feature subset sizes indicate very stable behavior (Figure 6).

5.5 Efficient Storage

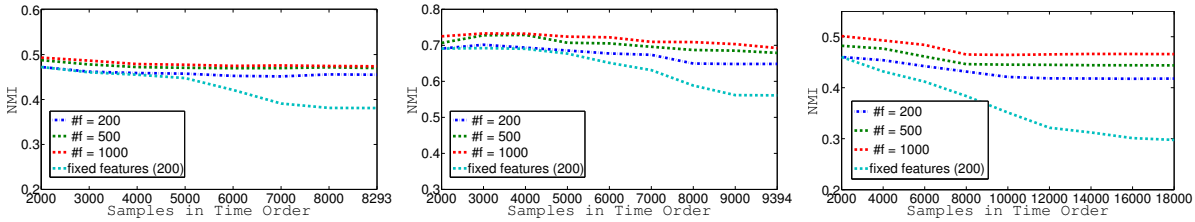
For many streaming applications, we not only want to identify the top-h features in the data but also want to store the data restricted to these top-h features at any time, to enable some further data mining analyses. In general, such analysis in streaming setting would require storing the whole data at each timestep as the set of the top-h features dynamically changes over time. However, using FSDS, we empirically noticed that storing the data stream restricted to top $g \times h$ features at each intermediate timestep, and a final selection of only those features which appear in the top $g \times h$ features in each of the intermediate timestep, suffices to get good results, even when g is a small number (i.e., 4). The results are shown in Figure 7. The number of features tested in these experiments are $h = \{200, 400, 600, 800, 1000\}$ with $g = \{1, 2, 3, 4\}$. We also report FSDS results where we



(a) Reuter21578 (b) TDT2 (c) 20Newsgroup (d) RCV1

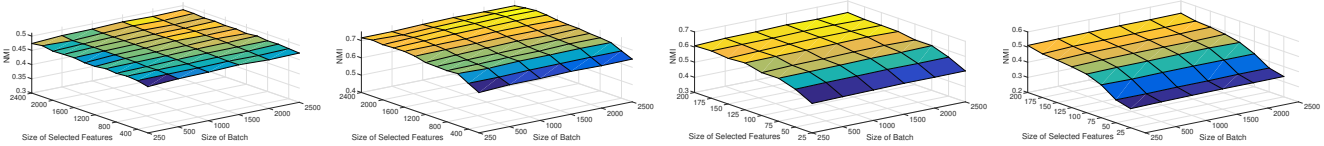
(e) USPS (f) MNIST (g) Tiny

Figure 3: Performance of the compared algorithms on different datasets. A randomly shuffled data stream is used for the online algorithms. Kmeans and StreamKM algorithms were executed on the whole feature set (hence they always have horizontal lines). For all other approaches, the quality of feature selection methods is measured by executing K-means clustering on the selected feature subset. For the RCV1 and Tiny datasets, MCFS and LaplacianScore experiments ran into memory issues as they require storing the quadratic sized affinity matrix. Memory issues also prevented the completion of GFW-p2 on Tiny. Our proposed approach (FSDS) achieves at least 97% NMI of MCFS while operating in a highly scalable streaming setting.



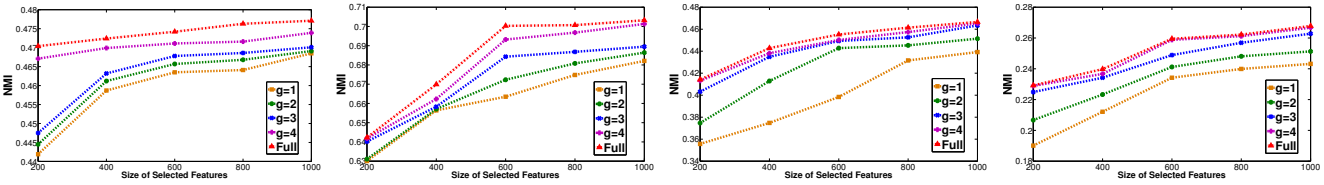
(a) Reuter21578 (b) TDT2 (c) 20Newsgroup

Figure 5: Concept drift test across time for FSDS. The results show that FSDS provides a stable performance even in presence of inherent concept drift in the data stream.



(a) Reuter21578 (b) TDT2 (c) USPS (d) MNIST

Figure 6: Stability test for FSDS across different batch sizes and feature subset sizes.



(a) Reuter21578 (b) TDT2 (c) 20Newsgroup (d) RCV1

Figure 7: Performance comparison of FSDS with different values of g.

store the entire data, and then use the data to obtain the final top h features (we call this strategy Full). Even setting g = 1, we already achieve about 91% NMI compared to the Full (FSDS) at h = 200. As we increase g, the results

get better and it suggests that it could be enough to store data restricted to the top $O(h)$ features at each timestep to enable further analyses.

6. CONCLUSION

We proposed an unsupervised feature selection algorithm for handling high dimensional data points arriving in a streaming fashion. Our algorithm uses ideas from matrix sketching to generate a continuous low-rank approximation of the input, which is then used in a regularized regression framework to obtain the individual feature weights. The algorithm only requires one-pass over the data, utilizes limited storage, and operates in near-real time. Theoretical results and experimental validation confirm that our proposed algorithm is efficient in both space and time for the task of streaming unsupervised feature selection.

7. REFERENCES

[1] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu. A framework for Projected Clustering of High Dimensional Data Streams. In *VLDB*, 2004.

[2] M. Belkin and P. Niyogi. Laplacian Eigenmaps and Spectral Techniques for Embedding and Clustering. In *NIPS*, volume 14, pages 585–591, 2001.

[3] D. Cai. Datasets. http://bit.ly/1IpMnAy.

[4] D. Cai, C. Zhang, and X. He. Unsupervised Feature Selection for Multi-cluster Data. *SIGKDD*, 2010.

[5] S. Chatterjee and A. S. Hadi. *Regression Analysis by Example*. John Wiley & Sons, 2013.

[6] A. Dasgupta, P. Drineas, B. Harb, V. Josifovski, and M. W. Mahoney. Feature Selection Methods for Text Classification. In *SIGKDD*, pages 230–239, 2007.

[7] M. Ghashami and J. M. Phillips. Relative Errors for Deterministic Low-Rank Matrix Approximations. In *SODA*, pages 707–717, 2014.

[8] G. H. Golub and C. F. Van Loan. *Matrix Computations*, volume 3. JHU Press, 2012.

[9] X. He, D. Cai, and P. Niyogi. Laplacian Score for Feature Selection. In *NIPS*, pages 507–514, 2005.

[10] C. Hou, F. Nie, D. Yi, and Y. Wu. Feature Selection via Joint Embedding Learning and Sparse Regression. *IJCAI*, 2011.

[11] H. Huang and S. Kasiviswanathan. Streaming Anomaly Detection Using Randomized Matrix Sketching. http://bit.ly/1FaDw6S.

[12] H. Huang, S. Yoo, D. Yu, and H. Qin. Diverse Power Iteration Embeddings and Its Applications. In *ICDM*, 2014.

[13] H. Huang, S. Yoo, D. Yu, and H. Qin. Noise-resistant Unsupervised Feature Selection via Multi-perspective Correlations. In *ICDM*, 2014.

[14] A. Krizhevsky, V. Nair, and G. Hinton. CIFAR-100 Dataset. http://bit.ly/1chapl0.

[15] H. Li, X. Wu, Z. Li, and W. Ding. Online Group Feature Selection from Feature Streams. In *AAAI*, 2013.

[16] Z. Li, Y. Yang, J. Liu, X. Zhou, and H. Lu. Unsupervised Feature Selection using Nonnegative Spectral Analysis. 2012.

[17] E. Liberty. Simple and Deterministic Matrix Sketching. In *SIGKDD*, pages 581–588, 2013.

[18] F. Lin and W. W. Cohen. A Very Fast Method for Clustering Big Text Datasets. In *ECAI*, 2010.

[19] C. Maung and H. Schweitzer. Pass-efficient Unsupervised Feature Selection. In *NIPS*, 2013.

[20] J. Misra and D. Gries. Finding Repeated Elements. *Science of computer programming*, 2(2), 1982.

[21] I. Ntoutsi, A. Zimek, T. Palpanas, P. Kröger, and H.-P. Kriegel. Density-based Projected Clustering over High Dimensional Data Streams. In *SDM*, 2012.

[22] S. Papadimitriou, J. Sun, and C. Faloutsos. Streaming Pattern Discovery in Multiple Time-series. In *VLDB*, 2005.

[23] S. Perkins and J. Theiler. Online Feature Selection Using Grafting. In *ICML*, 2003.

[24] A. Rahimi and B. Recht. Random Features for Large-scale Kernel Machines. In *NIPS*, 2007.

[25] M. Shindler, A. Wong, and A. W. Meyerson. Fast and Accurate k-means for Large Datasets. In *NIPS*, 2011.

[26] Q. Song, J. Ni, and G. Wang. A Fast Clustering-based Feature Subset Selection Algorithm for High-dimensional Data. *TKDE*, 25(1), 2013.

[27] A. Strehl and J. Ghosh. Cluster Ensembles–A Knowledge Reuse Framework for Combining Multiple Partitions. *JMLR*, 2003.

[28] A. Torralba, R. Fergus, and W. T. Freeman. 80 million Tiny Images: A Large Data Set for Nonparametric Object and Scene Recognition. *TPAMI*, 2008.

[29] J. Wang, P. Zhao, S. Hoi, and R. Jin. Online Feature Selection and its Applications. *TKDE*, 26(3), 2014.

[30] J. Weston, A. Elisseeff, B. Scholkopf, and M. Tipping. Use of the Zero Norm with Linear Models and Kernel Methods. *JMLR*, 2003.

[31] D. M. Witten and R. Tibshirani. A Framework for Feature Selection in Clustering. *JASA*, 2010.

[32] X. Wu, K. Yu, H. Wang, and W. Ding. Online Streaming Feature Selection. In *ICML*, 2010.

[33] S. Xiang, X. Shen, and J. Ye. Efficient Sparse Group Feature Selection via Nonconvex Optimization. In *ICML*, 2013.

[34] S. Xiang, T. Yang, and J. Ye. Simultaneous Feature and Feature Group Selection through Hard Thresholding. In *SIGKDD*, pages 532–541, 2014.

[35] H. Yang, M. R. Lyu, and I. King. Efficient Online Learning for Multitask Feature Selection. *TKDD*, 7(2):6, 2013.

[36] S. Yang, L. Yuan, Y. C. Lai, X. Shen, P. Wonka, and J. Ye. Feature Grouping and Selection over an Undirected Graph. In *SIGKDD*, 2012.

[37] Y. Yang, H. T. Shen, Z. Ma, Z. Huang, and X. Zhou. $l_{2,1}$-norm Regularized Discriminative Feature Selection for Unsupervised Learning. *IJCAI*, 2011.

[38] J. Zhou, D. P. Foster, R. A. Stine, and L. H. Ungar. Streamwise Feature Selection. *JMLR*, 7, 2006.

[39] X. Zhu, Z. Huang, Y. Yang, T. H. Shen, C. Xu, and J. Luo. Self-taught Dimensionality Reduction on the High-dimensional Small-sized Data. *Pattern Recognition*, 46(1), 2013.

[40] H. Zou and T. Hastie. Regularization and Variable Selection via the Elastic Net. *Journal of the Royal Statistical Society*, 67(2), 2005.